Logical Architecture and UIMA Analysis Engine Design & Implementation

Name: Qichen Pan   AndrewID: pqichen

Introduction

In this homework, I am required to design an analysis engine for simple Question & Answer test. The input data of the test model is a document that contains a question and several answers. This test model is supposed to give each answer a score by some algorithm. A certain number of answers with higher score are regarded as candidate answers. By evaluating the precision of these candidates, one can estimate the performance of whole system. In this task, a fixed UIMA type system is given, so I am supposed to develop my own codes based on the type system framework.

Type System Design

The annotation types are given in the type system, so my work is focusing on develop related analysis engines. Since there are a lot of types to be annotated, my engine system may have multiple components. So I develop several partial annotators along with their descriptor. And at the final stage, I come up with a Aggregate Analysis Engine to capsule all the other small analysis engines. The XML descriptors have a structure as shown in figure 2.1 below.

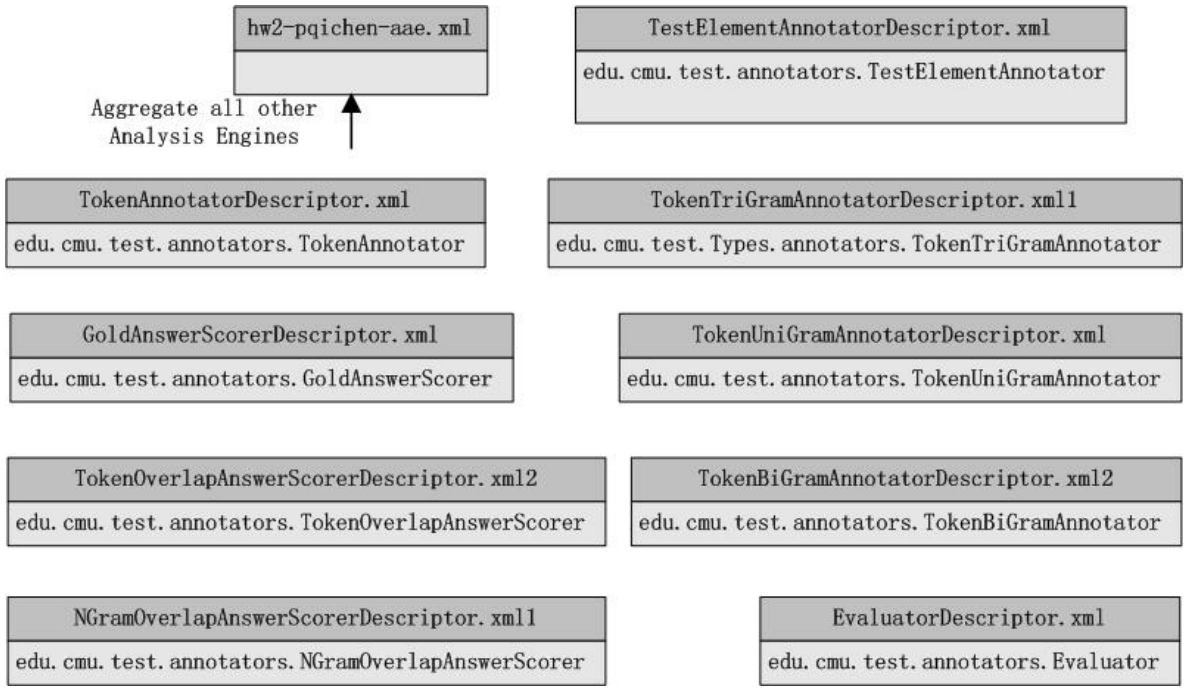
hw2-pqichen-aae.xml
Aggregate all other Analysis Engines
TestElementAnnotatorDescriptor.xml
edu.cmu.test.annotators.TestElementAnnotator
TokenAnnotatorDescriptor.xml
edu.cmu.test.annotators.TokenAnnotator
TokenTriGramAnnotatorDescriptor.xml1
edu.cmu.test.Types.annotators.TokenTriGramAnnotator
GoldAnswerScorerDescriptor.xml
edu.cmu.test.annotators.GoldAnswerScorer
TokenUniGramAnnotatorDescriptor.xml
edu.cmu.test.annotators.TokenUniGramAnnotator
TokenOverlapAnswerScorerDescriptor.xml2
edu.cmu.test.annotators.TokenOverlapAnswerScorer
TokenBiGramAnnotatorDescriptor.xml2
edu.cmu.test.annotators.TokenBiGramAnnotator
NGramOverlapAnswerScorerDescriptor.xml1
edu.cmu.test.annotators.NGramOverlapAnswerScorer
EvaluatorDescriptor.xml
edu.cmu.test.annotators.Evaluator

Figure 2.1 Descriptor structure

Analysis Engines Structure

Figure 3.1 is a simple graph which shows the structure of analysis engines, and the relationship between them.

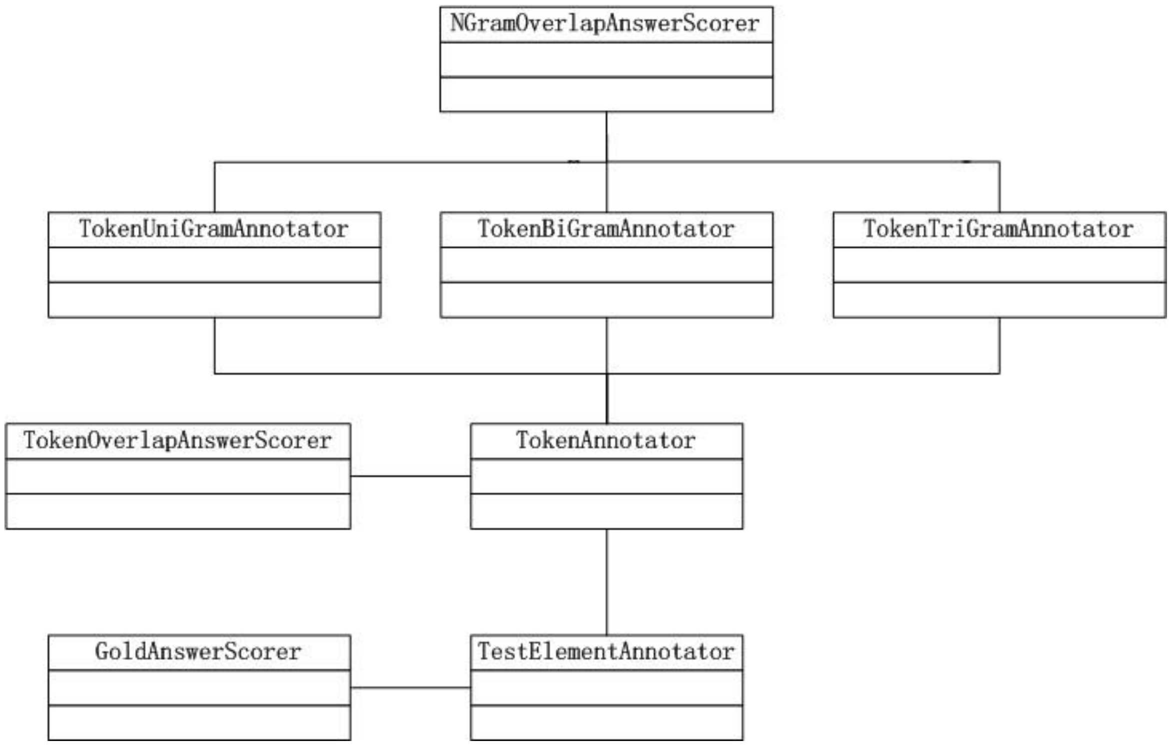

Figure 3.1 Analysis Engine Structure

- TestElmentAnnotator: This Annotator is responsible for annotating Questions and Answers. It maintains a Variable Qst to store Question information, for that each file only has one Question. It maintains an array to store Answers information. All operations are performed on DocumentsText.
- TokenAnnotator: This Annotator is responsible for annotating Tokens. It maintains an array to store Tokens information.Previous Question and Answers Annotations produced by TestElementAnnotator need to be retrieved.
- TokenUniGramAnnotator: This Annotator is responsible for annotating UniGrams. It maintains a array to store UniGrams information.Previous Tokens need to be retrieved. Actually, UniGrams are another form of Tokens. So I just transfer them from Tokens.
- TokenBiGramAnnotator: This Annotator is responsible for annotating BiGrams. It maintains a array to store BiGrams information. Previous Question and Answers need to be retrieved to calculate BiGram Number. Previous Tokens need to be retrieved. Pilot and Pilot_tk are designed to record array positions of BiGram and Token respectively.
- TokenTriGramAnnotator: This Annotator is responsible for annotating TriGrams. It maintains a array to store TriGrams information. Previous Question and Answers need to be retrieved to calculate TriGram Number. Previous Tokens need to be retrieved. Pilot and Pilot_tk are designed to record array positions of TriGram and Token respectively.

- GoldAnswerScorer: This Annotator is responsible for annotating AnswerScores in Gold Rule. It maintains a array to store Gold AnswerScore information. Previous Answers need to be retrieved. This algorithm simply assign 'true' to an answer if it is true. So Gold AnswerScore is guaranteed to have a 1.00 precision.
- TokenOverlapAnswerScorer: This Annotator is responsible for annotating AnswerScores in TokenOverlap Algorithm. It maintains a array to store TokenOverlap AnswerScore information. Previous Question and Answers need to be retrieved.This algorithm assign a score the same value as the hit ratio of AnswerTokens and QuestionTokens.
- NGramOverlapAnswerScorer: This Annotator is responsible for annotating AnswerScores in NGramOverlap Algorithm. It maintains a array to store NGramOverlap AnswerScore information. Previous Question and Answers need to be retrieved. Previous NGram need to be retrieved. This algorithm assign a score the same value as the hit ratio of AnswerNGrams and QuestionNGrams.
- Evaluator: This class is responsible for evaluating the performance of three algorithms. It first retrieves Gold_AS info to get the right answer number say 'n'. It then sorts the two other AnswerScore_List and picks the top n ones. It calculates the precision of three different algorithms and output them on the screen. The precisions are stored in precision_TokenOverlap, precision_NGramOverlap, precision_Gold.

Comparison of Different Design Methods

During the implementing process, I came up with some different ways of designing Analysis Engines.

- How to design sort algorithm for AnswerScore:
  In the first place, I decided to apply Arrays.sort() methods on AnswerScore objects. But the Constraint here was that the objects must be comparable. So I was thinking about making AnswerScore a comparable class by implementing the comparable interface. But the tutorial of this homework said that we are not allowed to modify the original type system so I did not use this solution.
  I defined another class named ‘AnswerScoreComparable’ to implement the comparable interface but eclipse kept on throwing error message stating that this class could not be loaded.
  So finally I realized that in our algorithm there was no need to develop a whole sort process. Instead I only need to find the n answer with largest score. So I wrote a binary search method to search the top AnswerScore in n loops. It also performs well.
- How to get Tokens when using TokenOverlap Algorithm:
  The first version of my design was to manually redo the same steps in TokenAnnotator when I need to find tokens in questions and answers. This scheme is very straight forwards for that with the question and answers annotations we can easily capture any tokens inside this range. And without loading any previous token information, we can

complete the whole calculation process with question and answers.
But the disadvantage is that by doing so I ignore the potential usage of former annotations. And also add some extra computational task to the whole system.
Another concern is that in the next NGramOverlap algorithm, if I already have a similar code design, I can reuse these codes immediately with only some trivial modification.

- How to get the final Score in NGramOverlap case:
  My first concern of this question was that there might be some weight for each NGram. Maybe BiGram plays a more important role than both UniGram an TriGram for that it can always present a subject-predicate relation. But in a short time I realized that my thought was naïve. Language is very complicated and there are plenty of flamboyant but meaningless phases. Another problem of assigning weights is that there are no more training data and also there is no approach to any test data. This method of getting the final score can be represented as shown below:

$$\mathrm{Score}(\mathrm{A_i}) = \alpha\frac{\mathrm{UniGram}(\mathrm{A_i}) \cap \mathrm{UniGram}(\mathrm{Q})}{\mathrm{UniGram}(\mathrm{A_i})} + \beta\frac{\mathrm{BiGram}(\mathrm{A_i}) \cap \mathrm{BiGram}(\mathrm{Q})}{\mathrm{BiGram}(\mathrm{A_i})} + \gamma\frac{\mathrm{TriGram}(\mathrm{A_i}) \cap \mathrm{TriGram}(\mathrm{Q})}{\mathrm{TriGram}(\mathrm{A_i})}$$

$$\alpha + \beta + \gamma = 1$$

  So finally I choose to use the method mentioned in instructor's slides. The equation below shows the way to get the final score:

$$\mathrm{Score}(\mathrm{A_i}) = \frac{\mathrm{UniGram}(\mathrm{A_i}) \cap \mathrm{UniGram}(\mathrm{Q}) + \mathrm{BiGram}(\mathrm{A_i}) \cap \mathrm{BiGram}(\mathrm{Q}) + \mathrm{TriGram}(\mathrm{A_i}) \cap \mathrm{TriGram}(\mathrm{Q})}{\mathrm{UniGram}(\mathrm{A_i}) + \mathrm{BiGram}(\mathrm{A_i}) + \mathrm{TriGram}(\mathrm{A_i})}$$

Summary

My Analysis Engine can work in a pipeline to perform as annotators of given type system. It can be used to analysis any simple Q&A files which contains a question and several answers.

The outputs of my system are precisions of GoldAnswerScorer, TokenOverlapAnswerScorer and NGramOverlapAnswerScoer respectively. They are stored in precision_TokenOverlap, precision_NGramOverlap and precision_Gold.

In the future I want to develop more Q&A cases to train my analysis engine and also to design some more sophisticated algorithm to assign final scores.